Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 6 (2011) 408–412

Procedia Computer Science

Complex Adaptive Systems, Volume 1
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2011- Chicago, IL

TOSSIM simulation of wireless sensor network serving as hardware platform for Hopfield neural net configured for max independent set

Jiakai Li, Gursel Serpen

Electrical Engineering and Computer Science, University of Toledo, Toledo, Ohio 43606, USA

Abstract

This paper, the third one in a three-paper sequence, presents the result of TOSSIM simulation of a Hopfield neural network as a static optimizer and configured to solve the maximum independent set (MIS) problem using a wireless sensor network as a fully parallel and distributed computing hardware platform. TinyOS with its default protocol stack along with nesC were used to develop the simulation model. Simulations were realized for mote counts of 10, 50, 100, and 182; messaging complexity, memory and simulation time costs were measured. Results indicated, as the most prominent finding, that the neural optimization algorithm was able to compute solutions to the MIS problem. The memory footprint of the TOSSIM process in Windows XP environment was about 20 MB for the range of sensor networks considered. The messaging complexity as measured by the total number of messages transmitted and the simulation time increased rather quickly indicating a need to optimize and tune certain aspects of the simulation environment if wireless sensor networks with higher mote counts need to be simulated.

Keywords: TOSSIM, nesC, TinyOS, wireless sensor network, Hopfield neural network, static optimization, parallel and distributed computation, message complexity, scalability

1. Introduction

This paper, which is the third one in a three-paper sequence, discusses the simulation study using the TOSSIM software tool for the nesC-TinyOS model of fully parallel and distributed computations of solutions of maximum independent set (MIS) problem through a Hopfield neural network embedded on a wireless sensor network which serves as the hardware platform. The first paper in the sequence [1] presented the maximum independent set problem definition, the Hopfield neural network computational model, and the mapping of Hopfield network to a wireless sensor network for fully parallel and distributed computation. The second paper in the sequence [2] discussed the TinyOS-nesC model for the Hopfield network configured for the MIS problem for fully parallel and distributed computation on a wireless sensor network.

The simulation study aims to validate the utility of a wireless sensor network as a parallel and distributed hardware platform for neural network computations, profile the cost in space, time and message complexity, and assess the scaling properties of the proposed computing paradigm which encompasses a neural network embedded within a sensor network for full parallelism and maximum distributed computation.

2. TinyOS-nesC and TOSSIM

The application code implementing Hopfield network configured for the maximum independent set problem was written using the nesC (network embedded systems C) programming language on the TinyOS operating system for wireless sensor networks [3]. TinyOS is open source with BSD-license, and is the most widely used operating system designed for distributed wireless sensor networks. nesC is a component-based, event-driven programming language used to build applications for the TinyOS platform. The nesC programming language is built as an extension to the C programming language with components "wired" together to run applications on TinyOS. TinyOS version 1.1.11.3 was used to implement this project on Windows XP OS with Cygwin user interface. TinyOS also requires the Java SDK: the default version of Java SDK (1.4.2) was utilized.

Default wireless sensor network protocols as incorporated into the TinyOS were used. TinyOS employs Berkeley-MAC (B-MAC) [4] medium access protocol which is CSMA based. To achieve low power consumption, B-MAC employs an adaptive preamble sampling scheme to reduce duty cycle and to minimize idle listening. TinyOS uses an ad-hoc multi-hop routing protocol, which leverages a shortest-path-first algorithm with a single destination node and active two-way link estimation. TinyOS employs the Flooding Time Synchronization Protocol (FTSP) for time synchronization, which is an ad-hoc, multi-hop time synchronization protocol for WSNs [5].

TOSSIM is a discrete event simulator, or more precisely emulator, of TinyOS at the bit level. An event is generated for each transmitted or received bit rather than one per packet. TOSSIM is a simulator for the execution of nesC model on TinyOS/MICA hardware. TOSSIM further works as an emulator of actual hardware through mapping hardware interrupts to discrete events. TOSSIM also has a built-in simulated radio model. In conclusion, TOSSIM is a high fidelity emulator of a network of TinyOS/MICA motes.

TOSSIM is conceived to study the behaviour of TinyOS and its applications: it is not intended to profile the performance of new protocols. Accordingly, there is one significant drawback and that is the fact that every mote must run the same code. Consequently, the usability of TOSSIM for applications involving heterogeneous sensors is limited at best. As such, TOSSIM is not appropriate for heterogeneous applications. TOSSIM has been shown to handle simulations of a sensor network with around a thousand motes. Its ability to simulate larger mote counts is primarily limited by its bit-level granularity: its performance degrades as the traffic increases. Channel sampling is also simulated at the bit level and consequently the use of a CSMA protocol causes significant overhead particularly when compared to a TDMA-based MAC protocol.

In TOSSIM the wireless communications is assumed to be error free for any mote within the one-hop neighborhood of a given mote. In other words, every bit transmitted in the network is received without any error. All signals are of equal strength, and collisions are modeled through the use of a CSMA protocol and there is no cancellation during network communication. This means that distance does not affect signal strength. Motes in the network may hear the overlap of the signals. The radio model is set as "empirical" based on an outdoor trace of packet connectivity with the RFM1000 radios. The transmission range for any node is the same, which is a disk with a radius of *r*.

TOSSIM makes the virtual location of each mote available to that mote through the “Location” interface. This is accomplished by setting the value of "fake" ADC channels on each mote, which the “FakeLocation” component reads to determine the mote's virtual location. This is meant to act as a stand-in for a real localization service when simulating TinyOS applications. A timer embedded in each mote may be used to implement the periodic activities like the sleep schedule.

3. Simulation Study

Simulation study used the default values for all protocols implemented into TinyOS 1.1.x. The sensor nodes or motes are assumed to be randomly and uniformly distributed on a plane. Each mote is assigned a unique ID. Numbering for mote IDs starts at 0 and increases by 1 for each additional mote in the network. There are two kinds of motes in the sensor network being used to implement the neural network, the supervisory (aka reference or gateway) mote and generic motes. Generic motes perform single neuron computations while the supervisory mote acts as a “supervisor” – it monitors, collects, processes and broadcasts the global information related to neural network computations. The supervisory mote may also act as a generic mote if necessary. The supervisory mote may have different hardware design with practically unlimited power and may be able to continuously operate for extended periods of time which would be unusual for typical generic WSN motes. The supervisory mote is responsible broadcasting start and end times of neural computation phases including initialization, convergence or neural network dynamics update, solution detection and analysis, and re-initialization as needed.

Neural network computations take place while the mote is awake and is subjected to the schedule of the mote on which the neuron resides. In this case, we use a timer on every node to manage timing during the network updating. The timers are set to fire independently or asynchronously by a time interval for all aspects of the WSN operation including neighborhood discovery, initialization, and periodic activity in normal operation. Each mote wakes up from sleep once every 1 time period (which is set to 1000 ms) through its dedicated timer. Timers are not synchronized globally: every timer operates independent of others and yet has the same wakeup frequency of 1 time period. Accordingly, each neuron output gets updated once per 1 time period asynchronously of other neuron outputs.

The maximum independent set (MIS) problem entails computing a maximum cardinality subgraph of a given graph such that there is no edge between any two vertices in the maximum cardinality subgraph. The motes in the wireless sensor network (WSN) correspond to vertices of the graph while one-hop communication links between motes correspond to edges between the corresponding vertices of the graph. In other words, if any two motes are within one-hop neighborhood of each other (assuming communication in both directions exists), then this communication link points at existence of an edge between the two corresponding vertices associated with those two motes. Accordingly, each mote in the WSN (a vertex of the corresponding graph) must know its one-hop neighborhood or its connection topology. The connection topology information of each mote (which maps to a neuron in the neural network or vertex in the graph) is needed to be able to compute the weights or the weight vector for a given neuron. Each mote stores its one-hop neighborhood (adjacency or connectivity information) and weight vector for its neuron local memory. The first step in the initialization is the identification of the one-hop neighbors of each mote. Therefore, it is necessary to implement the so-called "neighborhood discovery" phase. When the neighborhood discovery begins, every mote sends out its own mote ID information through a so-called beacon packet as specified in the TinyOS environment. Motes are randomly selected for broadcasting based on the TinyOS B-MAC CSMA protocol. Each mote must broadcast its beacon packet and broadcast only once during the neighborhood discovery period. After broadcasting its beacon packet, each mote also sends a so-called "report message" to the supervisory mote to indicate that it has broadcasted the beacon packet. Those who have already broadcasted may not send more beacon messages. Upon receipt of a beacon packet with a mote ID, a mote will store the one-hop neighbor's address from an incoming message in local memory and mark it as its neighbor. The supervisory mote keeps a tally to store the report messages from the generic motes. If the report message tally indicates that all motes reported having sent the beacon message, the supervisory mote will broadcast a control message to the entire network and tell everyone that the neighborhood discovery ended.

The supervisory mote manages the phases of neural computation. It will determine the convergence of the neural optimization algorithm, identify if the converged state is a solution, and communicate the start and end times of all such phases to the generic motes. After the initialization phase is complete, every neuron holds its own initial randomly-generated output value, its weight vector and the bias value in the local memory of the associated mote. Upon entry to the relaxation and convergence phase, motes wake up per the schedule of their timers and update the output value of its neuron and broadcast the newly-updated neuron output value to its one-hop neighbors.

The simulations were performed for WSNs with mote counts of 10, 50, 100, and 182. Each mote count case was simulated 10 times and a student distribution was used to statistically characterize the data. The settings for the two constraint weight parameters in conjunction with the bound derived in the second paper in the three-paper sequence [2] are presented in Table 1. Three performance metrics were measured: memory footprint of the TOSSIM process as indicated by the Windows™ OS, the number of packets exchanged during the neural computation, and the total simulation time for the TOSSIM process as provided by the TOSSIM itself. The total number of packets is measured by summing all the messages sent out by each mote which has a local counter to keep track of all the messages transmitted.

Simulation results are presented in Tables 2 through 4, and Figures 1 and 2 as box-and-whiskers plots. One immediate observation is that the memory footprint of the TOSSIM process is hardly affected by the size of the WSN (as measured by the number of motes): the difference is approximately 1.7 MB between the footprints of TOSSIM processes for the 10-mote (19.5 MB) and the 182-mote (21.2 MB) WSN instances. However the message complexity as measured by the total number of packets transmitted by the sensor network increases dramatically from tens of thousands for the 10-mote instance to several millions for the 182-mote instance. The simulation time is also similarly affected: it takes around 20 seconds to simulate the 10-mote sensor network, while the time increases to nearly 3 minutes for the 182-mote network case. For all three complexity measures (memory space, message, and simulation time), t-distribution parameter values are presented in Tables 2, 3, and 4, respectively.

Mote Count / **Weight Parameters**	**10**	**50**	**100**	**182**
g_a	0.3	0.5	0.6	0.6
g_b	1.0	1.0	1.0	1.0

Table 1. Value settings for constraint weight parameters for Hopfield neural network.

	Mote Count			
	10	**50**	**100**	**182**
Mean (packets)	22725.6	41127	527596	3842638
Deviation	23737.36	43275.25	42634.70	2707150.00

Table 2. t-distribution model parameter values for message complexity measure

	Mote Count			
	10	**50**	**100**	**182**
Mean (Seconds)	23.058	21.848	41.228	100.604
Deviation	9.015	7.393	7.369	40.956

Table 3. t-distribution model parameter values for time complexity measure

	Mote Count			
	10	**50**	**100**	**182**
Mean (KB)	19523	19679	20090	21215
Deviation	2.797	1.647	10.163	11.12

Table 4. t-distribution model parameter values for space complexity (TOSSIM process memory space) measure

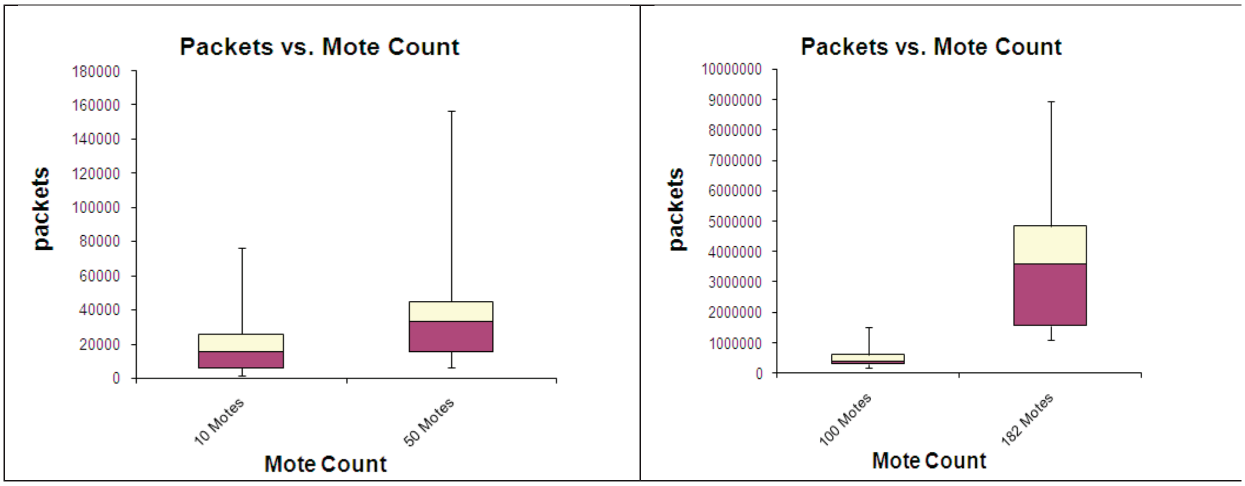

Figure 1. Messaging cost for different mote counts.

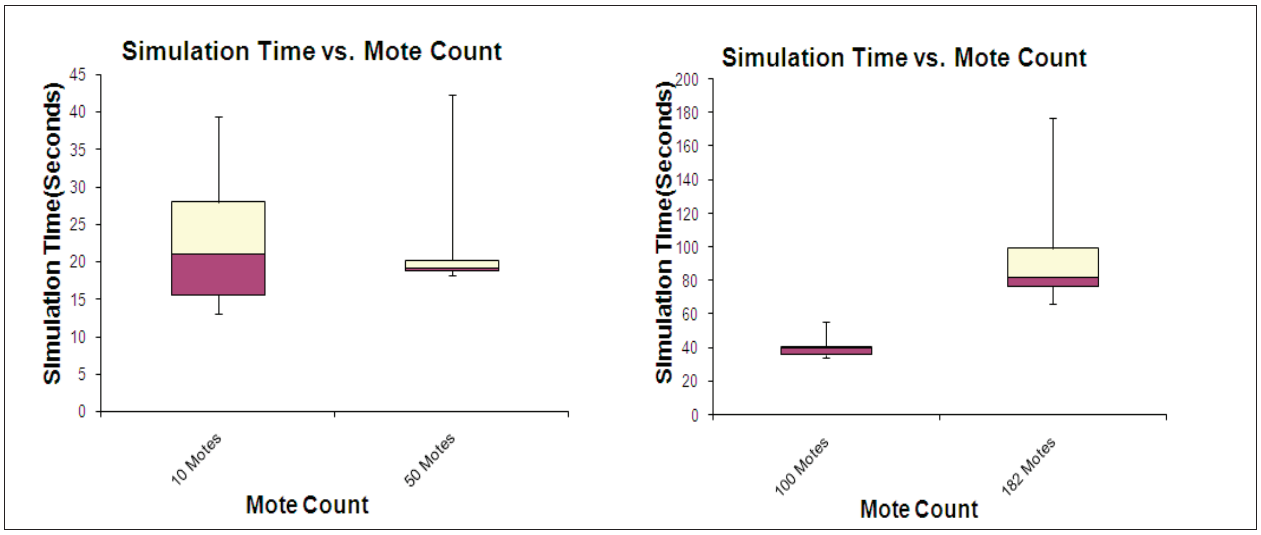

Figure 2. Simulation time for different mote counts.

Hopfield network dynamics may demonstrate substantial variations in terms of computational effort to accomplish the convergence to a fixed point. This variation is fundamentally affected by the random initial values of neuron outputs, and the random update order of neurons. As a result significant variations in the messaging cost and the simulation time as presented in Figures 1 through 2 are expected.

The CSMA based MAC protocol and the rather primitive routing protocol employed did play a relatively important role in the apparently high cost of messaging and simulation time. This is also a point of improvement with the inclusion of more efficient protocols for the follow-up simulations. We therefore expect the messaging cost and simulation times to dramatically decrease upon leveraging a more efficient protocol stack for the network.

4. Conclusions

This paper presented the simulation results, using TOSSIM, of a neural optimization algorithm being executed in fully parallel and distributed mode on a wireless sensor network as the hardware computing platform for the maximum independent set problem. Simulation results indicated that the proposed parallel and distributed computation of neural optimization problem is feasible. The complexity analysis through the simulation study showed that it is necessary to address the cost of messaging and extended simulation times for large scale sensor networks through incorporation of more sophisticated MAC and routing protocols in the TinyOS environment. We further plan to use an actual wireless sensor network composed of Mica2 motes [6] to validate the application program and profile its performance particularly in terms of message complexity and execution time.

References

[1] Serpen, G. and Li, J., “Parallel and distributed computations of maximum independent set by Hopfield neural net embedded into a wireless sensor network”, Proc. of Complex Adaptive Systems Conference, Chicago, 2011.

[2] Li, J. and Serpen, G., “A nesC-TinyOS model for parallel and distributed computations of max independent set by Hopfield network on a wireless sensor network”, Proc. of Complex Adaptive Systems Conference, Chicago, 2011.

[3] Gay,D., Levis, P. Behren, R., Welsh, M., Brewer, E. and Culler.D “The nesC language: A holistic approach to networked embedded systems” *in Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation* ACM New York, NY, USA 2003.

[4] Polastre, J., Hill, J. and Culler, D. "Versatile Low Power Media Access for Wireless Sensor Networks" *in Proceeding of SenSys '04 Proceedings of the 2nd international conference on embedded networked sensor systems,* New York, NY, USA 2004.

[5] Maróti, M., Kusy, B., Simon, G. and Ledeczi, A. “The flooding time synchronization protocol” *in Proceedings of SenSys '04 Proceedings of the 2nd international conference on embedded networked sensor systems,* New York, NY, USA 2004.

[6] MEMSIC Inc., Mica 2 sensor board at http://www memsic.com/products/wireless-sensor-networks/wireless-modules html, cited 30 May 2011.